2º curso / 2º cuatr.

Grado Ing. Inform.

Doble Grado Ing. Inform. y Mat.

Arquitectura de Computadores (AC)

Cuaderno de prácticas.
Bloque Práctico 1. Programación paralela I: Directivas OpenMP

Estudiante (nombre y apellidos): Alberto Jesús Durán López
Grupo de prácticas: 1
Fecha de entrega:
Fecha evaluación en clase:

Recuerde que debe adjuntar al zip de entrega, el pdf de este fichero, todos los ficheros con código fuente implementados/utilizados y el resto de ficheros que haya implementado/utilizado (scripts, hojas de cálculo, etc.), lea la Sección 1.4 del guion]

Ejercicios basados en los ejemplos del seminario práctico

1. Usar la directiva `parallel` combinada con directivas de trabajo compartido en los ejemplos `bucle-for.c` y `sections.c` del seminario. Incorporar el código fuente resultante al cuaderno de prácticas.

RESPUESTA: código fuente `bucle-forModificado.c`

```
/* Tipo de letra Courier New o Liberation Mono. Tamaño 8 o 9.*/
/* COPIAR Y PEGAR CÓDIGO FUENTE AQUÍ*/
/* INTERLINEADO SENCILLO */

#include <stdio.h>
#include <stdlib.h>
#include <omp.h>

int main(int argc, char ** argv)
{
int main(int argc, char **argv) {

            int i, n = 9;
            if(argc < 2) {
                        fprintf(stderr,"\n[ERROR] - Falta no
iteraciones \n");
                        exit(-1);
            }

            n = atoi(argv[1]);

            #pragma omp parallel for
                        for (i=0; i<n; i++){
                                    printf("thread %d ejecuta la
iteración %d del bucle\n", omp_get_thread_num(),i);
                        }

            return(0);

}
```

RESPUESTA: código fuente `sectionsModificado.c`

```
/* Tipo de letra Courier New o Liberation Mono. Tamaño 8 o 9.*/
/* COPIAR Y PEGAR CÓDIGO FUENTE AQUÍ*/
/* INTERLINEADO SENCILLO */

#include <stdio.h>
#include <stdlib.h>
#include <omp.h>

int main(int argc, char ** argv)
{
void funcA() {
            printf("En funcA: esta sección la ejecuta el thread%d\n",
omp_get_thread_num());
}

void funcB() {
            printf("En funcB: esta sección la ejecuta el thread%d\n",
omp_get_thread_num());
}

int main() {

            #pragma omp parallel sections
            {
                        #pragma omp section
                        (void) funcA();

                        #pragma omp section
                        (void) funcB();
            }

}
```

2. Imprimir los resultados del programa `single.c` usando una directiva `single` dentro de la construcción `parallel` en lugar de imprimirlos fuera de la región `parallel`. Añadir lo necesario, dentro de la nueva directiva `single` incorporada, para que se imprima el identificador del thread que ejecuta el bloque estructurado de la directiva `single`. Incorpore en su cuaderno de trabajo el código fuente y volcados de pantalla con los resultados de ejecución obtenidos.

   RESPUESTA: código fuente `singleModificado.c`

```
/* Tipo de letra Courier New o Liberation Mono. Tamaño 8 o 9.*/
/* COPIAR Y PEGAR CÓDIGO FUENTE AQUÍ*/
/* INTERLINEADO SENCILLO */

#include <stdio.h>
#include <stdlib.h>
#include <omp.h>

int main(int argc, char ** argv)
{
int n = 9, i, a, b[n];


            for (i=0; i<n; i++) b[i] = -1;


            #pragma omp parallel
            {
                        #pragma omp single
                        {
                                    printf("\nIntroduce valor de
inicialización a: ");
                                    scanf("%d", &a );
                                    printf("\nSingle ejecutada por el
thread %d\n", omp_get_thread_num());
                        }


                        #pragma omp for
                        for (i=0; i<n; i++)
                                    b[i] = a;


                        #pragma omp single
                                    printf("Salida:\n");
                                    for(i=0; i<n; i++)
                                                printf("b[%d] =
%d\t",i,b[i]);
                                    printf("\n");
                                    printf("Single ejecutada por el
thread %d,\n", omp_get_thread_num());
            }
            /*
            printf("Después de la región parallel:\n");


            for (i=0; i<n; i++) printf("b[%d] = %d\t",i,b[i]);
                        printf("\n");
            */
}
```

CAPTURAS DE PANTALLA:

```
albduranlopez@albduranlopez:~/Escritorio/Segunda Entrega AC/código$ gcc -O2 -fopenmp single-Modificado.c -o arch
albduranlopez@albduranlopez:~/Escritorio/Segunda Entrega AC/código$ ./arch

Introduce valor de inicialización a: 20

Single ejecutada por el thread 5
Salida:
b[0] = 20       b[0] = 20       b[0] = 20       b[1] = 20       b[2] = 20       b[3] = 20       b[4] = 20       b[5] = 20b
[6] = 20        b[7] = 20       b[8] = 20
Single ejecutada por el thread 7,
b[0] = 20
Single ejecutada por el thread 5,

Single ejecutada por el thread 2,
b[0] = 20
Single ejecutada por el thread 0,

Single ejecutada por el thread 4,
b[0] = 20
Single ejecutada por el thread 1,
b[0] = 20
Single ejecutada por el thread 6,
b[0] = 20
Single ejecutada por el thread 3,
```

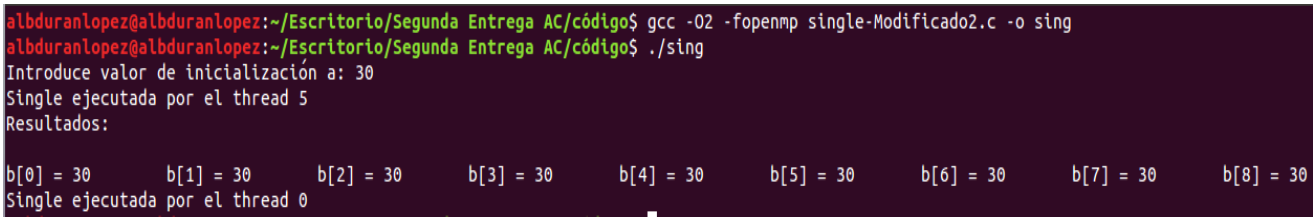
3. Imprimir los resultados del programa `single.c` usando una directiva `master` dentro de la construcción `parallel` en lugar de imprimirlos fuera de la región `parallel`. Añadir lo necesario, dentro de la nueva directiva `master` incorporada, para que se imprima el identificador del thread que ejecuta el bloque estructurado de la directiva `master`. Incorpore en su cuaderno el código fuente y volcados de pantalla con los resultados de ejecución obtenidos. ¿Qué diferencia observa con respecto a los resultados de ejecución del ejercicio anterior?

   RESPUESTA: código fuente `singleModificado2.c`

```
/* Tipo de letra Courier New o Liberation Mono. Tamaño 8 o 9.*/
/* COPIAR Y PEGAR CÓDIGO FUENTE AQUÍ*/
/* INTERLINEADO SENCILLO */

#include <stdio.h>
#include <stdlib.h>
#include <omp.h>

int main(int argc, char ** argv)
{           int n = 9, i, a, b[n];


            for (i=0; i<n; i++)
                        b[i] = -1;


            #pragma omp parallel
            {
                        #pragma omp single
                        {
                                    printf("Introduce valor de
inicialización a: ");
                                    scanf("%d", &a );
                                    printf("Single ejecutada por el thread
%d\n", omp_get_thread_num());
                        }


                        #pragma omp for
                        for (i=0; i<n; i++)
```

```
                                        b[i] = a;


                                #pragma omp master
                                {
                                        printf("Resultados:\n\n");
                                        for (i=0; i<n; i++)
                                                printf(" b[%d] = %d\t",
i , b[i]);
                                        printf("\nSingle ejecutada por el
thread %d\n", omp_get_thread_num());
                                }
                        }


}
```

CAPTURAS DE PANTALLA:



RESPUESTA A LA PREGUNTA: Ejecutamos varias veces el código y comprobamos que la directiva es siempre ejecutada por el thread 0 (como observamos en la última línea de la captura)

4. ¿Por qué si se elimina directiva `barrier` en el ejemplo `master.c` la suma que se calcula e imprime no siempre es correcta? Responda razonadamente.

   RESPUESTA: Porque la directiva barrier hace que en una determinada parte del código los threads se esperen entre sí, por lo que al eliminar dicha directiva, el thread 0 se ejecutaría más rápido, dándose lugar a que no se haya realizado la suma correcta y el resultado sea incorrecto.

1.1.1

Resto de ejercicios

5. El programa secuencial C del Listado 1 calcula la suma de dos vectores (v3 = v1 + v2; v3(i) = v1(i) + v2(i), i=0,…N-1). Generar el ejecutable del programa del Listado 1 para **vectores globales**. Usar `time` (Lección 3/ Tema 1) en la línea de comandos para obtener, en el PC local, el tiempo de ejecución (*elapsed time*) y el tiempo de CPU del usuario y del sistema generado. Obtenga los tiempos para vectores con 10000000 componentes. ¿La suma de los tiempos de CPU del usuario y del sistema es mayor o igual que el tiempo real (*elapsed*)? Justifique la respuesta.

CAPTURAS DE PANTALLA:

Como usamos vectores globales descomentamos “#define VECTOR_GLOBAL”

```
albduranlopez@albduranlopez:~/Escritorio$ gcc -O2 SumaVectores.c -o suma -lrt
albduranlopez@albduranlopez:~/Escritorio$ time ./suma 10000000
Tiempo(seg.):0.022707188         / Tamaño Vectores:10000000     / V1[0]+V2[0]=V3
[0](1000000.000000+1000000.000000=2000000.000000) / / V1[9999999]+V2[9999999]=V3
[9999999](1999999.900000+0.100000=2000000.000000) /

real    0m0.052s
user    0m0.040s
sys     0m0.012s
```

La suma de los tiempos de CPU del usuario y del sistema es igual al tiempo real (elapsed) ya que únicamente se ejecuta un núcleo del procesador

6. Generar el código ensamblador a partir del programa secuencial C del Listado 1 para **vectores globales** (para generar el código ensamblador tiene que compilar usando `-S` en lugar de `-o`). Utilice el fichero con el código fuente ensamblador generado y el fichero ejecutable generado en el ejercicio 5 para obtener para atcgrid los MIPS (*Millions of Instructions Per Second*) y los MFLOPS (*Millions of FLOating-point Per Second*) del código que obtiene la suma de vectores (código entre las funciones `clock_gettime()`); el cálculo se debe hacer para 10 y 10000000 componentes en los vectores (consulte la Lección 3/Tema1 AC). Incorpore **el código ensamblador de la parte de la suma de vectores** en el cuaderno.

   CAPTURAS DE PANTALLA:

```
[E1estudiante8@atcgrid ~]$ echo './suma 10' | qsub -q ac
53101.atcgrid
[E1estudiante8@atcgrid ~]$ ls -lag
total 180
drwx------    6 E1estudiante8  4096 mar 28 18:48 .
drwxr-xr-x. 500 root          20480 feb 15 13:42 ..
-rw-rw-r--    1 E1estudiante8   764 mar  3 12:54 archivo
-rw-------    1 E1estudiante8  1932 mar  2  2016 .bash_history
-rw-r--r--    1 E1estudiante8    18 ene 16  2015 .bash_logout
-rw-r--r--    1 E1estudiante8   193 ene 16  2015 .bash_profile
-rw-r--r--    1 E1estudiante8   231 ene 16  2015 .bashrc
-rwxrwxr-x    1 E1estudiante8  8944 mar  7 19:03 dinam
-rwxrwxr-x    1 E1estudiante8  8928 mar  7 18:53 global
drwxrwxr-x    2 E1estudiante8  4096 mar  6 19:47 hello
drwxr-xr-x    3 E1estudiante8  4096 feb 25  2015 .local
drwxr-xr-x    4 E1estudiante8  4096 ene 30  2015 .mozilla
drwxr-xr-x    2 E1estudiante8  4096 feb  5  2015 .ssh
-rw-------    1 E1estudiante8     0 mar  3 12:55 STDIN.e43318
-rw-------    1 E1estudiante8     0 mar  7 17:54 STDIN.e45427
-rw-------    1 E1estudiante8    93 mar  7 18:14 STDIN.e45445
-rw-------    1 E1estudiante8   979 mar  7 18:17 STDIN.e45450
-rw-------    1 E1estudiante8   832 mar  7 18:18 STDIN.e45453
-rw-------    1 E1estudiante8   832 mar  7 18:20 STDIN.e45455
-rw-------    1 E1estudiante8     0 mar 28 18:48 STDIN.e53101
-rw-------    1 E1estudiante8 21366 mar  3 12:55 STDIN.o43318
-rw-------    1 E1estudiante8   153 mar  7 17:54 STDIN.o45427
-rw-------    1 E1estudiante8     0 mar  7 18:14 STDIN.o45445
-rw-------    1 E1estudiante8   468 mar  7 18:17 STDIN.o45450
-rw-------    1 E1estudiante8  1023 mar  7 18:18 STDIN.o45453
-rw-------    1 E1estudiante8  1023 mar  7 18:20 STDIN.o45455
-rw-------    1 E1estudiante8   148 mar 28 18:48 STDIN.o53101
-rwxrwxr-x    1 E1estudiante8  8866 mar 28 18:47 suma
-rwxrwxr-x    1 E1estudiante8  3311 mar  9 11:07 SumaVectores.c
-rwxrwxr-x    1 E1estudiante8  8848 mar  7 18:20 SumaVectoresC
-rw-rw-r--    1 E1estudiante8  2853 mar 28 18:47 SumaVectores.s
-rwxrwxrwx    1 E1estudiante8   744 mar  7 18:10 SumaVectores.sh
[E1estudiante8@atcgrid ~]$ cat STDIN.o53101
Tiempo(seg.):0.000002428         / Tamaño Vectores:10   / V1[0]+V2[0]=V3[0](1.00
0000+1.000000=2.000000) / / V1[9]+V2[9]=V3[9](1.900000+0.100000=2.000000) /
[E1estudiante8@atcgrid ~]$
```

```
[E1estudiante8@atcgrid ~]$ cat STDIN.o53110
Tiempo(seg.):0.048020210        / Tamaño Vectores:10000000     / V1[0]+V2[0]=V3[0](1000
000.000000+1000000.000000=2000000.000000) / / V1[9999999]+V2[9999999]=V3[9999999](199999
9.900000+0.100000=2000000.000000) /
```

RESPUESTA: Cálculo de los MIPS y los MFLOPS

Para un tamaño del vector de 10 componentes, se obtiene un tiempo de 0.000002428 segundos.

En ensamblador tenemos 7 instrucciones fuera del bucle y 6 dentro por lo que 10*6 = 60.

En total tenemos 60+7=67 instrucciones que tardan el tiempo indicado anteriormente, por lo que:

$$\text{MIPS} = \frac{67}{0.000002428 * 10^6} = 27{,}5947 \text{ mips}$$

De estas 67 instrucciones, 30 son en coma flotante:

$$\text{MFLOPS} = \frac{30}{0.000002428 * 10^6} = 12{,}3558 \text{ mflops}$$

Para un tamaño del vector de 10 millones de componentes,se obtiene un tiempo de 0.048020210 segundos

Aplicamos el mismo razonamiento anterior y como resultado obtenemos:

$$\text{MIPS} = \frac{60.000.007}{0.048020210 * 10^6} = 1249.4740 \text{ mips}$$

$$\text{MFLOPS} = \frac{30.000.000}{0.048020210 * 10^6} = 624.7369 \text{ mflops}$$

RESPUESTA:

Código ensamblador generado de la parte de la suma de vectores

```
            call        clock_gettime
            xorl        %eax, %eax
            .p2align 4,,10
            .p2align 3
.L5:
            movsd       v1(%rax), %xmm0
            addq        $8, %rax
            addsd       v2-8(%rax), %xmm0
            movsd       %xmm0, v3-8(%rax)
            cmpq        %rbx, %rax
            jne         .L5
.L6:
            leaq        16(%rsp), %rsi
            xorl        %edi, %edi
            call        clock_gettime
```

7. Implementar un programa en C con OpenMP, a partir del código del Listado 1, que calcule en paralelo la suma de dos vectores (v3 = v1 + v2; v3(i)=v1(i)+v2(i), i=0,…N-1) usando las directivas `parallel` y `for`. Se debe paralelizar también las tareas asociadas a la inicialización de los vectores. Como en el código del Listado 1 se debe obtener el tiempo (*elapsed time*) que supone el cálculo de la suma. Para obtener este tiempo usar la función `omp_get_wtime()`, que proporciona el estándar OpenMP, en lugar de `clock_gettime()`. NOTAS: (1) el número de componentes N de los vectores debe ser un argumento de entrada al programa; (2) se deben inicializar los vectores antes del cálculo; (3) se debe asegurar que el programa calcula la suma correctamente imprimiendo todos los componentes del vector resultante, v3, para varios tamaños pequeños de los vectores (por ejemplo, N = 8 y N=11); (5) se debe imprimir sea cual sea el tamaño de los vectores el tiempo de ejecución del código paralelo que suma los vectores y, al menos, el primer y último componente de v1, v2 y v3 (esto último evita que las optimizaciones del compilador eliminen el código de la suma).

RESPUESTA: código fuente implementado

```
#include <stdlib.h> //atoi(), malloc() y free()
#include <stdio.h> //printf()
#include <time.h> //clock_gettime()
#include <omp.h>//biblioteca omp


#define PRINTF_ALL // comentar para quitar el printf
#define MAX 33554432


double v1[MAX], v2[MAX], v3[MAX];
int main(int argc, char** argv) {
    int i;
    //Diferencia de tiempo entre el inicio y el final
    double dif;
    double inicio, final;


    if (argc<2) {
        printf("Faltan indicar componentes del vector\n");
        exit(-1);
    }


    unsigned int num = atoi(argv[1]); // Máximo N =2^32-1


    #ifdef VECTOR_GLOBAL
    if (num>MAX)
      num=MAX;
             #endif


    //Inicializar vectores
    #pragma omp parallel for
    for(i=0; i<num; i++) {
        v1[i] = num*0.1+i*0.1;
        v2[i] = num*0.1-i*0.1;
    }


    inicio=omp_get_wtime();
    //Calcular suma de vectores
```

```
    #pragma omp parallel for
    for(i=0; i<num; i++)
        v3[i] = v1[i] + v2[i];


    final=omp_get_wtime();
    //Calculamos el tiempo (end-start)
    dif=final-inicio;


    //Imprimir resultado de la suma y el tiempo de ejecución
    #ifdef PRINTF_ALL
    printf("Tiempo(seg.):%11.9f\t / Tamaño Vectores:%u\n",dif,num);
    for(i=0; i<num; i++)
        printf("/ V1[%d]+V2[%d]=V3[%d](%8.6f+%8.6f=%8.6f) /\n",
i,i,i,v1[i],v2[i],v3[i]);
              #else
    printf("Tiempo(seg.):%11.9f\t / Tamaño Vectores:%u\t/
V1[0]+V2[0]=V3[0](%8.6f+%8.6f=%8.6f) / /V1[%d]+V2[%d]=V3[%d](%8.6f+
%8.6f=%8.6f) /\n",dif,num,v1[0],v2[0],v3[0],num-1,num-1,num-1,v1[num-
1],v2[num-1],v3[num-1]);
              #endif


    return 0;
}
```

(RECUERDE ADJUNTAR CÓDIGO FUENTE AL .ZIP)

CAPTURAS DE PANTALLA (compilación y ejecución para N=8 y N=11):

8. Implementar un programa en C con OpenMP, a partir del código del Listado 1, que calcule en paralelo la suma de dos vectores usando las `parallel` y `sections/section` (se debe aprovechar el paralelismo de datos usando estas directivas en lugar de la directiva `for`); es decir, hay que repartir el trabajo (tareas) entre varios threads usando `sections/section`. Se debe paralelizar también las tareas asociadas a la inicialización de los vectores. Para obtener este tiempo usar la función `omp_get_wtime()` en lugar de `clock_gettime()`. NOTAS: (1) el número de componentes N de los vectores debe ser un argumento de entrada al programa; (2) se deben inicializar los vectores antes del cálculo; (3) se debe asegurar que el programa calcula la suma correctamente imprimiendo todos los componentes del vector resultante, v3, para tamaños pequeños de los vectores (por ejemplo, N = 8 y N=11); (5) se debe imprimir sea cual sea el tamaño de los vectores el tiempo de ejecución del código paralelo que suma los vectores y, al menos, el primer y último componente de v1, v2 y v3 (esto último evita que las optimizaciones del compilador eliminen el código de la suma).

RESPUESTA: código fuente implementado

```
#include <stdlib.h> //atoi(), malloc() y free()
#include <stdio.h> //printf()
#include <time.h> //clock_gettime()
#include <omp.h>//biblioteca omp


#define PRINTF_ALL // comentar para quitar el printf ...
#define MAX 33554432 //=2^25


double v1[MAX], v2[MAX], v3[MAX];
int main(int argc, char** argv) {
    int i;
    double ncgt, cgt1,cgt2;




    if (argc<2) {
        printf("Faltan no componentes del vector\n");
        exit(-1);
    }
    unsigned int N = atoi(argv[1]); // Máximo N =2^32-1
#ifdef VECTOR_GLOBAL
    if (N>MAX) N=MAX;
#endif


    //Inicializar vectores
        #pragma omp parallel private(i)
        {
                #pragma omp sections
                {
                        #pragma omp section
                        for(i=0; i<N/4; i++)
                        {
                                v1[i] = N*0.1+i*0.1;
                                v2[i] = N*0.1-i*0.1;
                        }


                        #pragma omp section
                        for(i=N/4; i<N/2; i++)
```

```
                        {
                                v1[i] = N*0.1+i*0.1;
                                v2[i] = N*0.1-i*0.1;
                        }


                        #pragma omp section
                        for(i=N/2; i<3*N/4; i++)
                        {
                                v1[i] = N*0.1+i*0.1;
                                v2[i] = N*0.1-i*0.1;
                        }


                        #pragma omp section
                        for(i=3*N/4; i<N; i++)
                        {
                                v1[i] = N*0.1+i*0.1;
                                v2[i] = N*0.1-i*0.1;
                        }
                }


                #pragma omp single
                {
                        cgt1 = omp_get_wtime();
                }


                //Calcular suma de vectores
                #pragma omp sections
                {
                        // Dividimos las iteraciones for de forma
manual en 4 pedazos
                        #pragma omp section
                        for(i=0; i<N/4; i++)
                                v3[i] = v1[i] + v2[i];


                        #pragma omp section
                        for(i=N/4; i<N/2; i++)
                                v3[i] = v1[i] + v2[i];


                        #pragma omp section
                        for(i=N/2; i<3*N/4; i++)
                                v3[i] = v1[i] + v2[i];

                        #pragma omp section
                        for(i=3*N/4; i<N; i++)
                                v3[i] = v1[i] + v2[i];


                }


                #pragma omp single
```

```
                {
                        cgt2 = omp_get_wtime();
                }
        }
        ncgt = cgt2-cgt1;


    //Imprimir resultado de la suma y el tiempo de ejecución
#ifdef PRINTF_ALL
    printf("Tiempo(seg.):%11.9f\t / Tamaño Vectores:%u\n",ncgt,N);
    for(i=0; i<N; i++)
        printf("/ V1[%d]+V2[%d]=V3[%d](%8.6f+%8.6f=%8.6f)
/\n",i,i,i,v1[i],v2[i],v3[i]);
#else
    printf("Tiempo(seg.):%11.9f\t / Tamaño Vectores:%u\t/
V1[0]+V2[0]=V3[0](%8.6f+%8.6f=%8.6f) / /V1[%d]+V2[%d]=V3[%d](%8.6f+
%8.6f=%8.6f) /\n",ncgt,N,v1[0],v2[0],v3[0],N-1,N-1,N-1,v1[N-1],v2[N-
1],v3[N-1]);
#endif


    return 0;
}
```

(RECUERDE ADJUNTAR CÓDIGO FUENTE AL .ZIP)

CAPTURAS DE PANTALLA (compilación y ejecución para N=8 y N=11):

```
albduranlopez@albduranlopez:~/Escritorio$ gcc -O2 -fopenmp SumaVectores8.c -o sumaejer8 -lrt
albduranlopez@albduranlopez:~/Escritorio$ ./sumaejer8 8
Tiempo(seg.):0.003664763         / Tamaño Vectores:8
/ V1[0]+V2[0]=V3[0](0.800000+0.800000=1.600000) /
/ V1[1]+V2[1]=V3[1](0.900000+0.700000=1.600000) /
/ V1[2]+V2[2]=V3[2](1.000000+0.600000=1.600000) /
/ V1[3]+V2[3]=V3[3](1.100000+0.500000=1.600000) /
/ V1[4]+V2[4]=V3[4](1.200000+0.400000=1.600000) /
/ V1[5]+V2[5]=V3[5](1.300000+0.300000=1.600000) /
/ V1[6]+V2[6]=V3[6](1.400000+0.200000=1.600000) /
/ V1[7]+V2[7]=V3[7](1.500000+0.100000=1.600000) /
albduranlopez@albduranlopez:~/Escritorio$ ./sumaejer8 11
Tiempo(seg.):0.000540507         / Tamaño Vectores:11
/ V1[0]+V2[0]=V3[0](1.100000+1.100000=2.200000) /
/ V1[1]+V2[1]=V3[1](1.200000+1.000000=2.200000) /
/ V1[2]+V2[2]=V3[2](1.300000+0.900000=2.200000) /
/ V1[3]+V2[3]=V3[3](1.400000+0.800000=2.200000) /
/ V1[4]+V2[4]=V3[4](1.500000+0.700000=2.200000) /
/ V1[5]+V2[5]=V3[5](1.600000+0.600000=2.200000) /
/ V1[6]+V2[6]=V3[6](1.700000+0.500000=2.200000) /
/ V1[7]+V2[7]=V3[7](1.800000+0.400000=2.200000) /
/ V1[8]+V2[8]=V3[8](1.900000+0.300000=2.200000) /
/ V1[9]+V2[9]=V3[9](2.000000+0.200000=2.200000) /
/ V1[10]+V2[10]=V3[10](2.100000+0.100000=2.200000) /
albduranlopez@albduranlopez:~/Escritorio$
```

9. ¿Cuántos threads y cuántos cores como máximo podría utilizar la versión que ha implementado en el ejercicio 7? Razone su respuesta. ¿Cuántos threads y cuantos cores como máximo podría utilizar la versión que ha implementado en el ejercicio 8? Razone su respuesta.

RESPUESTA:

En el ejercicio 7, la constante OMP_NUM_THREADS indica las hebras que crea la directiva for, que aunque se correspondan con los cores del PC, se pueden cambiar. Pero como no hemos definido esa variable de entorno, se usarán todos los cores/threads disponibles.

Por otro lado, en el ejercicio 8 hemos dividido el bucle en 4 partes, se crearán 4 hebras, 1 por cada parte del bucle.

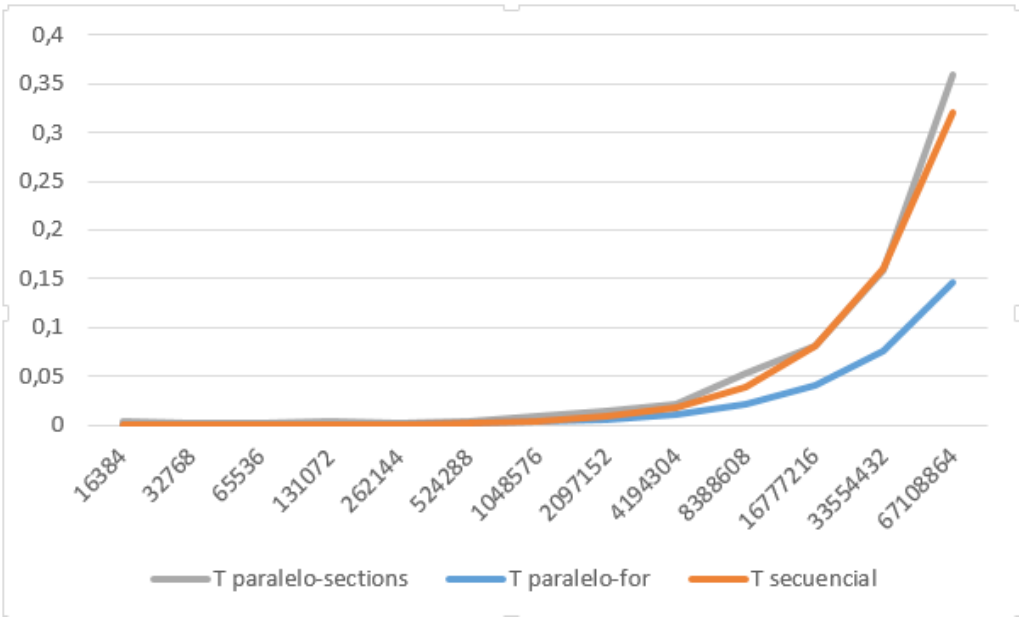
10. Rellenar una tabla como la Tabla 2 para atcgrid y otra para el PC local con los tiempos de ejecución de los programas paralelos implementados en los ejercicios 7 y 8 y el programa secuencial del Listado 1. Generar los ejecutables usando -O2. En la tabla debe aparecer el tiempo de ejecución del trozo de código que realiza la suma en paralelo (este es el tiempo que deben imprimir los programas). Ponga en la tabla el número de threads/cores que usan los códigos. Represente en una gráfica los tres tiempos. NOTA: Nunca ejecute en atcgrid código que imprima todos los componentes del resultado.

RESPUESTA:

ATCGRID

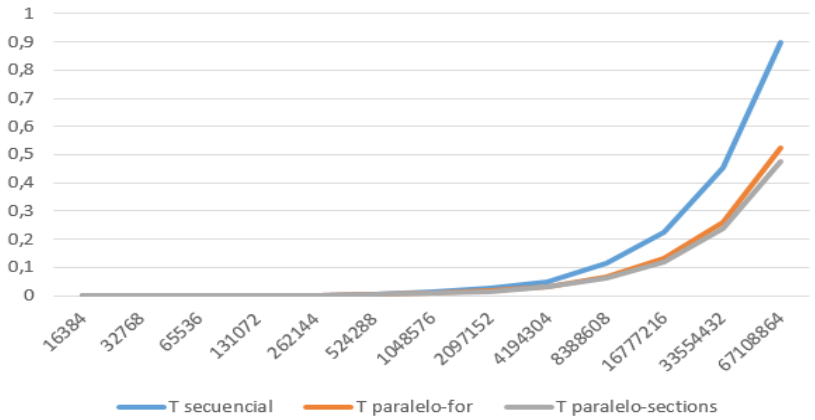
Nº de Componentes	**T. secuencial vect. Globales** **1 thread/core**	**T. paralelo (versión `for`)** **24 threads/cores**	**T. paralelo (versión `sections`)** **24 threads/cores**
16384	0,000062997	0,000065851	0.003961940
32768	0,000125994	0,000125116	0.001158264
65536	0,000239389	0,000237721	0.001685220
131072	0,000502717	0,00045167	0.004413319
262144	0,000955162	0,000858173	0.001875220
524288	0,002005841	0,001630529	0.004213319
1048576	0,004212266	0,003098005	0.008551180
2097152	0,008845758	0,005886209	0.013721302
4194304	0,018576091	0,011183798	0.020430588
8388608	0,039009792	0,0212492161	0.052896519
16777216	0,081145258	0,040373511	0.081039414
33554432	0,160491716	0,076709671	0.159022917
67108864	0.320783432	0.145748375	0.358664011

s y de las dos



MI PC (Los tiempos no salen completos ya que a la hora de pasarlos de la tabla del excel se truncan. En el excel incluído aparecen los tiempos completos)

Nº de Componentes	T. secuencial vect. Dinámicos 1 thread/core	T. paralelo (versión `for`) 4 threads/cores	T. paralelo (versión `sections`) 4 threads/cores
16384	0,00134	0,00013	0,00013
32768	0,00025	0,00025	0,00026
65536	0,00051	0,00051	0,00052
131072	0,00111	0,00192	0,00106
262144	0,00242	0,00194	0,0021
524288	0,00504	0,00409	0,0042
1048576	0,0121	0,00818	0,00844
2097152	0,02717	0,01636	0,0149
4194304	0,05093	0,03272	0,0298
8388608	0,11313	0,06543	0,06159
16777216	0,22594	0,13087	0,1182
33554432	0,45292	0,26173	0,23839
67108864	0,89673	0,52346	0,47677



11. Rellenar una tabla como la Tabla 3 para el PC local con el tiempo de ejecución, tiempo de CPU del usuario y tiempo CPU del sistema obtenidos con `time` para el ejecutable del ejercicio 7 y para el programa secuencial del Listado 1. Ponga en la tabla el número de threads/cores que usan los códigos. ¿El tiempo de CPU que se obtiene es mayor o igual que el tiempo real (*elapsed*)? Justifique la respuesta.

RESPUESTA:

En el programa secuencial, el tiempo de CPU coincide con el tiempo real ya que únicamente se usa un procesador.
En el programa paralelo, el tiempo de CPU se calcula sumando el tiempo de cada ciclo y el tiempo real es el tiempo que tarda el programa en ejecutarse por lo que el tiempo de CPU es mayor que el tiempo real.

Tabla 3. Tiempos de ejecución de la versión secuencial de la suma de vectores y de las dos versiones paralelas. Sustituir en el encabezado de la tabla "¿?" por el número de threads utilizados.

Nº de Componentes	Tiempo secuencial vect. Globales 1 thread/core			Tiempo paralelo/versión for 4 Threads/cores		
	Elapsed	*CPU-user*	*CPU- sys*	*Elapsed*	*CPU-user*	*CPU- sys*
65536	real 0m0.006s	user 0m0.002s	sys 0m0.004s	real 0m0.012s	user 0m0.052s	sys 0m0.004s
131072	real 0m0.009s	user 0m0.005s	sys 0m0.004s	real 0m0.009s	user 0m0.008s	sys 0m0.024s
262144	real 0m0.016s	user 0m0.007s	sys 0m0.008s	real 0m0.030s	user 0m0.108s	sys 0m0.032s
524288	real 0m0.028s	user 0m0.014s	sys 0m0.014s	real 0m0.012s	user 0m0.036s	sys 0m0.012s
1048576	real 0m0.054s	user 0m0.027s	sys 0m0.026s	real 0m0.029s	user 0m0.136s	sys 0m0.004s
2097152	real 0m0.106s	user 0m0.048s	sys 0m0.057s	real 0m0.019s	user 0m0.068s	sys 0m0.024s
4194304	real 0m0.210s	user 0m0.101s	sys 0m0.106s	real 0m0.042s	user 0m0.208s	sys 0m0.036s
8388608	real 0m0.408s	user 0m0.197s	sys 0m0.209s	real 0m0.072s	user 0m0.288s	sys 0m0.072s
16777216	real 0m0.824s	user 0m0.423s	sys 0m0.398s	real 0m0.140s	user 0m0.532s	sys 0m0.192s
33554432	real 0m1.639s	user 0m0.862s	sys 0m0.769s	real 0m0.300s	user 0m1.108s	sys 0m0.360s
67108864	real 0m2.167s	user 0m1.50s	sys 0m0.921s	real 0m0.608s	user 0m1.012s	sys 0m1.056s